A Multi-mode RCPSP with Stochastic Nonrenewable Resource Consumption

Muller, Laurent Flindt

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

A Multi-mode RCPSP with Stochastic Nonrenewable Resource Consumption

Report 4.2011

DTU Management Engineering

Laurent Flint Muller
March 2011

DTU Management Engineering
Department of Management Engineering

A Multi-mode Resource-Constrained Project Scheduling Problem with Stochastic Nonrenewable Resource Consumption

Laurent Flindt Muller

Department of Management Engineering, Technical University of Denmark
Produktionstorvet, Building 426, DK-2800 Kgs. Lyngby, Denmark
lafm@man.dtu.dk

Abstract

Many processes within production scheduling and project management involve the scheduling of a number of activities, each activity having a certain duration and requiring a certain amount of limited resources. The duration and resource requirements of activities are commonly the result of estimations, and thus generally subject to uncertainty. If this uncertainty is not taken into account the resulting schedules may not be robust in the sense that, when executed, the uncertainty may cause the schedules to take longer than expected, consume more resources, or be outright infeasible. We propose a new variant of the Multi-mode Resource-Constrained Project Scheduling Problem, where the nonrenewable resource requirements of each mode is given by a Gaussian distribution, and the nonrenewable resource constraints must be satisfied with a certain probability ϵ. Such constraints are also known as chance constraints. We present a Conic Quadratic Integer Program model of the problem, and describe and experiment with a branch-and-cut algorithm for solving the problem. In each node of the branch-and-bound tree, the branching decisions are propagated in order to remove variables from the problem, and thus improve bounds. In addition we experiment with cutting on the conic quadratic resource constraints. Computational experiments show that the branch-and-cut algorithm outperforms CPLEX 12.1. We finally examine the "cost of uncertainty" by investigating the relation between values of ϵ, the makespan, and the solution time.

1 Introduction

Many processes within production scheduling and project management involve the scheduling of a number of activities, each activity having a certain duration and requiring a certain amount of limited resources. Resources could for instance be machines, labor, budget, or stock. Precedence relations may exist between activities, such that one activity can not start before others are completed. Typically one wishes to schedule the activities such that the total time taken to complete them all is minimized. Being models of real-life processes, the duration and resource requirements of an activity is commonly the result of estimations, and thus generally subject to uncertainty. If this uncertainty is not taken into account the resulting schedules may not be robust in the sense that, when executed, the uncertainty may cause the schedules to take longer than expected, consume more resources, or be outright infeasible. Thus it is of interest to take into account uncertainty when scheduling such projects.

Disregarding uncertainty, the above described problem may be modeled as a Resource-Constrained Project Scheduling Problem (RCPSP), which is a generalization of the well-known Job-Shop-Scheduling problem. There exists a number of variants of the RCPSP, see for instance Blażewicz et al. (1983), Brucker et al. (1999), or Hartmann and Briskorn (2010). The Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP) is a popular variant, in which each activity can be performed in a number of different so-called modes, each representing alternative ways of executing the activity, i.e., different duration and resource requirements. A mode may for instance model that consuming more resources the activity will be completed faster, or that

there is an alternative for the choice of resources to be used. For the MRCPSP one distinguishes between renewable resources and nonrenewable resources: renewable resources are replenished in each timestep, while this is not the case for nonrenewable resources, here resource usage is accumulated across the entire project. A renewable resource could be man-hours per day, or the capacity of a machine, while a nonrenewable resource could be a budget, or some kind of stock.

Much research has focused on extending the RCPSP to the case where the activity durations are stochastic. There are different approaches: one approach is to construct a so-called policy, or strategy, which, when executed, will result in the best expected makespan across a number of scenarios. For policy-related results see Radermacher (1981), Igelmund and Radermacher (1983b), Möhring et al. (1984), Möhring et al. (1985), Möhring and Radermacher (1985), Radermacher (1986), Fernandez and Armacost (1996), and Fernandez et al. (1998a,b), and for computational results see Igelmund and Radermacher (1983a), Golenko-Ginzburg and Gonik (1997), Tsai and D Gemmill (1998), Valls et al. (1998), and Stork (2001). A different approach focuses on creating a so-called robust baseline schedule, which has a good probability of coping with uncertainty in activity durations, see for instance Chtourou and Haouari (2008) where a two-stage heuristic approach is developed, and Zhu et al. (2007) where a variant where the activities are additionally subject to due dates is considered, and a two-stage heuristic and an exact solution approach presented. A third approach focuses on disruption management, see for instance Van de Vonder et al. (2005), Van de Vonder (2006), Van de Vonder et al. (2006), Van de Vonder et al. (2007a), and Van de Vonder et al. (2007b, 2008). Uncertainty in the renewable resource requirements and capacities may also affect the quality of schedules, and some research has considered this case, see for instance Lambrechts et al. (2008a,b). No research seems to have examined the case of stochastic nonrenewable resource requirements.

Considering the case of stochastic nonrenewable resource requirements is of interest for at least two reasons: (1) Ideally one would like to be able to model and solve problems with uncertainty in activity durations, renewable and nonrenewable resource consumption, and renewable and nonrenewable resource capacities. The case considered here can be seen as another step in the direction of achieving this goal. (2) The problem may also be of interest in itself. Many projects span many years, and for some, uncertainty in cost may be of more interest, than, say, uncertainty in activity durations. Certainly many large projects are known to run over budget.

We thus consider a new variant of the RCPSP, where the nonrenewable resource requirements are stochastic, and where given a so-called risk factor $0 \leq \epsilon \leq 1$, one wishes to find a minimum makespan schedule, such that the nonrenewable resource constraints are satisfied with with probability at least ϵ. For modelling reason we will assume $\epsilon \geq 0.5$, which seems acceptable, since for practical applications it is hard to image scenarios where one would be interested in satisfying a budget with a propability less than 50%. We call this problem the Multi-mode Resource-Constrained Project Scheduling Problem with Stochastic Nonrenewable Resource Consumption (MRCPSP-SNR).

The contribution of this paper, is the description and modeling (as a Conic Quadratic Integer Program (CQIP)) of the MRCPSP-SNR. We propose and experiment with a branch-and-cut algorithm for solving the problem. In each node of the branch-and-bound tree, branching decisions are propagated, in a way similar to Constraint Programming (CP), in order to fix additional variables and improve bounds. The propagation is dependent on the variables of the model, and we experiment with adding additional (but redundant) variables to enable stronger branching. Computational experiments on a large set of instances, adapted from instances found in the literature, show that the branch-and-cut algorithm with propagation outperforms ILOG CPLEX 12.1 (CPLEX). We finally examine the "cost of uncertainty", that is: what is the effect of the risk factor on the makespan?

In Section 2 a formal description of the MRCPSP-SNR is given, in Section 3 the problem is modeled as a CQIP, in Section 4 cuts based on the second-order cone constraints are presented, in Section 5 the developed branch-and-cut procedure is described, includinf bounding arguments, and propagation rules, and in Section 6 we present the computational results. Finally we present the conclusion in Section 7.

2 Problem

We here give a formal description of the MRCPSP-SNR: A project consists of a set $\mathcal{A} = \{1, \ldots, n\}$ of *activities* to be scheduled. Traditionally, activity 1 and n are so-called dummy activities, which represent the start and the end of the project. Let $\mathcal{A}^* = \mathcal{A} \backslash \{1, n\}$. Each activity j can be performed in a number of different *modes* $\mathcal{M}_j = \{1, \ldots, |\mathcal{M}_j|\}$, each representing a different way of performing the activity. Let $\mathcal{T}$ be the set of time-steps during which the activities must be performed. There are two sets of resources, (1) *renewable resources* $\mathcal{R} = \{1, \ldots, |\mathcal{R}|\}$, and (2) *nonrenewable resources* $\tilde{\mathcal{R}} = \{1, \ldots, |\tilde{\mathcal{R}}|\}$. A renewable resource $k \in \mathcal{R}$, has capacity R_k in each time step, while a nonrenewable resource $k \in \tilde{\mathcal{R}}$ has capacity $\tilde{R}_k$ across all time steps of the project. When an activity j is scheduled in mode $m \in \mathcal{M}_j$, it has a *processing time* of p_{jm} (non-preemtible) and requires $r_{jkm} \geq 0$ units of renewable resource $k \in \mathcal{R}$ in each time period and an amount of nonrenewable for each resource $k \in \tilde{\mathcal{R}}$ across all time periods. The nonrenewable resource consumption is for each activity $j \in \mathcal{A}$ and each mode $m \in \mathcal{M}_j$ given by a Gaussian distribution, where $\tilde{r}_{jkm}$ is the mean, and σ^2_{jkm} is the variance. There exists *precedence relations* between the activities, such that one activity $j \in \mathcal{A}$ cannot be started before all its predecessors, $\mathcal{P}_j$, are completed. Symmetrically, $\mathcal{S}_j$ denotes the set of successors and $\mathcal{N}_j = \mathcal{A} \backslash (\mathcal{P}_j \cup \mathcal{S}_j)$ denotes the set of activities which can run in parallel with j. Let $\mathcal{E} = \{(i, j) \in \mathcal{A} \times \mathcal{A} : i \in \mathcal{P}_j\}$ be the set of all precedence relations. In addition to the project a risk factor $\epsilon \geq 0.5$ is given, and the objective is to find a precedence and resource-capacity feasible schedule which minimizes the makespan and satisfy the nonrenewable resource constraints with probability at least ϵ. The MRCPSP-SNR may be formulated as follows:

$$
\begin{aligned}
&\min \ \sigma_n \\
&s.t. \ \sigma_j \geq \sigma_i + p_{i,m(i)} && \forall (i,j) \in \mathcal{E} && (1)\\
&\quad \sum_{j \in A(t)} r_{j,k,m(j)} \leq R_k && \forall k \in \mathcal{R}, \forall t \in \mathcal{T} && (2)\\
&\quad \mathbf{P}\left(\left\{ \ \textstyle\sum_{j \in \mathcal{A}} \tilde{r}_{j,k,m(j)} \leq \tilde{R}_k, \quad \forall k \in \tilde{\mathcal{R}} \ \right\} \right) \geq \epsilon && && (3)\\
&\quad \sigma_j \geq 0 && \forall j \in \mathcal{A}, && (4)
\end{aligned}
$$

where σ_j is the starting time of activity j, $m(j)$ is mode chosen for activity j, and $A(t) = \{j \in \mathcal{A} | \sigma_j \leq t \leq \sigma_j + \pi_j\}$, i.e., the activities in progress at time t. Constraints (1) are denoted the *precedence constraints*, Constraints (2) are denoted the *renewable resource constraints* (or *resource constraints* for short), and Constraints (3) are denoted the *stochastic nonrenewable resource constraints*. These constraints are so-called chance constraints (see for instance Boyd and Vandenberghe (2004)), which means that the constraints must be satisfied with a certain probability ϵ. As mentioned previously, the RCPSP is a generalization of the Job Shop Scheduling Problem and is therefore $\mathcal{NP}$-hard (see e.g. Blażewicz et al. (1983)). As the MRCPSP-SNR reduces to the RCPSP when the variance off all activities are 0, and $\epsilon = 1$, the MRCPSP-SNR is also $\mathcal{NP}$-hard.

3 Model

In this section we first give a brief description of how a chance constraint may be modelled as a second-order cone constraint, next we propose how the joint chance constraints of the problem, i.e., the nonrenewable resource constraints, can be divided into independent chance constraints, and finally we show how the problem may be formulated as a CQIP.

3.1 Modelling chance constraints

A second-order cone constraint is a constraint of the form

$$ay + \omega \|Dy + d\|_2 \leq b,$$

where $a \in \Re^n$, $d \in \Re^n$, $b \in \Re$, $\omega \geq 0$ and $D \in \Re^{n \times n}$ are constants, $y \in \Re^n$ is a decision-variable, and $\| \cdot \|_2$ is the euclidean norm. A chance constraint of the form

$$\mathbf{P}(ry \leq s) \geq \epsilon,$$

where $s \in \Re$ and $0 \leq \epsilon \leq 1$ are a constants, and r is a Gaussian random vector, with mean vector μ and variance vector σ^2, may be formulated as the constraint (see e.g. Boyd and Vandenberghe (2004)):

$$\sum_{i=0}^{n} \mu_i y_i + \Phi^{-1}(\epsilon)\sqrt{\sigma_i^2 y_i^2} \leq b, \tag{5}$$

where where Φ is the cumulative distribution function. If $\epsilon \geq 0.5$, and thus $\Phi^{-1}(\epsilon) \geq 0$, then Constraint (5) is equivalent to a second-order cone constraint with $a = \mu$, $d = 0$, $\omega = \Phi^{-1}(\epsilon)$, $b = s$, and $D_{ii} = \sigma_i$, $D_{ij} = 0$ for $i \neq j$.

3.2 Separating joint chance constraints

The Constraints (3) are joint chance constraints, meaning that all of them must be satisfied with probability ϵ jointly. One can thus not simply reformulate each constraint individually as second-order cone constraints using the value of ϵ. Since in many cases $|\tilde{\mathcal{R}}|$ is a relatively small number, we propose to separate them into individual chance constraints as follows: Let ϵ be the given risk factor: $\forall k \in \tilde{\mathcal{R}}$ select $0.5 \leq \epsilon_k \leq 1 : \Pi_{k \in \tilde{\mathcal{R}}} \epsilon_k = \epsilon$, e.g., $\epsilon_k = \sqrt[|\tilde{\mathcal{R}}|]{\epsilon}$. The joint chance constraints may now be split into individual chance constraints using the selected values for each ϵ_k, and then formulated as second-order cone constraints.

3.3 Conic Quadratic Integer Program

For some activity i and mode $m \in \mathcal{M}_i$, let $\mathcal{T}_{im}^e$ and $\mathcal{T}_{im}^l$ be respectively the earliest and latest possible completion time of i when executed in mode m. $\mathcal{T}_{im}^e$ and $\mathcal{T}_{im}^l$ may be calculated given an upper bound on the makespan, and using any bound argument, such as the well-known critical path lower bound. Given a risk factor $\epsilon \geq 0.5$, the MRCPSP-SNR may be formulated as the following CQIP:

$$\min \sum_{m \in \mathcal{M}_n} \sum_{t=\mathcal{T}_{nm}^e}^{\mathcal{T}_{nm}^l} t \cdot x_{ntm} \tag{6}$$

$$s.t. \sum_{m \in \mathcal{M}_i} \sum_{t=\mathcal{T}_{im}^e}^{\mathcal{T}_{im}^l} t \cdot x_{itm} \leq \sum_{m \in \mathcal{M}_j} \sum_{t=\mathcal{T}_{jm}^e}^{\mathcal{T}_{jm}^l} x_{jtm}(t - p_{jm}) \qquad \forall (i,j) \in \mathcal{E} \tag{7}$$

$$\sum_{j \in \mathcal{A}} \sum_{m \in \mathcal{M}_j} \sum_{t'=\max\{t, \mathcal{T}_{jm}^e\}}^{\min\{t+p_{jm}-1, \mathcal{T}_{jm}^l\}} r_{jkm} \cdot x_{jt'm} \leq R_k, \qquad \forall k \in \mathcal{R}, \forall t \in \mathcal{T} \tag{8}$$

$$\sum_{j \in \mathcal{A}} \sum_{m \in M_j} \mu_{jkm} \cdot y_{jm} + \Phi^{-1}(\epsilon_k)\sqrt{\sum_{j \in \mathcal{A}} \sum_{m \in M_j} \sigma_{jkm}^2 \cdot y_{jm}} \leq \overline{R}_k, \qquad \forall k \in \tilde{\mathcal{R}} \tag{9}$$

$$\sum_{m \in \mathcal{M}_j} \sum_{t=\mathcal{T}_{jm}^e}^{\mathcal{T}_{jm}^l} x_{jtm} = 1, \qquad \forall j \in \mathcal{A} \tag{10}$$

$$x_{jtm} \in \{0,1\} \qquad \forall j \in \mathcal{A}, \forall m \in \mathcal{M}_j \tag{11}$$

where the binary variable x_{jtm} is 1 if and only if activity j completes at time t using mode m. For ease of exposition we have introduced artificial variables $y_{jm} := \sum_{t=\mathcal{T}_{jm}^e}^{\mathcal{T}_{jm}^l} x_{jtm}$, such that y_{jm} is 1 if and only if activity j is scheduled using mode m. We will in Section 6 investigate whether

it is worth including the y-variables explicitly in the model for the sake of branching. The *precedence constraints* (7) ensure that no activity is started before its predecessors have completed, the *renewable resource constraints* (8) ensure that at no point in time the resource requirements of the activities in progress exceed the capacity of a renewable resource, the *stochastic nonrenewable resource constraints* (9) ensure that the combined nonrenewable resource consumption of all activities satisfy the capacity of nonrenewable resources with probability at least ϵ, and the constraints (10) ensure that each activity is completed only once using exactly one mode. Note, that we have used the fact that $y_{jm}^2 = y_{jm}$ since y_{jm} is binary.

4 Conic cover cuts

We will in this section describe a class of cuts, which may be used to strengthen the formulation. As the proposed algorithm makes use of CPLEX, which is already effective at separating cuts for linear Mixed Integer Programming (MIP) constraints, we instead focus on cuts for the second-order cone constraints.

Let N be a finite index set and let $Q_1, \ldots, Q_{|K|}$ be a division of N into $|K|$ independent sets. i.e., $\bigcup_{k \in K} Q_k = N$, and $Q_i \cap Q_j = \emptyset, \forall i, j \in K, i \neq j$. Consider the following polytope

$$Z := \left\{ z \in \{0,1\}^{|N|} : \sum_{i=0}^{n} \mu_i z_i + \omega \sqrt{\sigma_i^2 z_i} \leq b, \sum_{i \in Q_k} z_i \leq 1, \forall k \in K \right\},$$

where $b \geq 0$ and $\omega \geq 0$. If $\omega = 0$, Z can be recognized as the classic knapsack polytope with Generalized Upper Bound (GUB) constraints. We thus refer to Z (when $\omega > 0$) as a second-order cone knapsack polytope with GUB constraints.

So-called cover cuts are well-known for the classic knapsack polytope: Given a set $C \subseteq N$ for which $\sum_{i \in C} \mu_i > b$ a cover cut has the form:

$$\sum_{i \in C} \mu_i \leq |C| - 1. \tag{12}$$

Given a cover C, the corresponding cover inequality may be strengthened by including additional variables. When chosen appropriately, these variables will add to the left-hand side of (12) without raising the right-hand side. The process of adding variables to an existing cover, is called *extending* the cover, and can be seen as a lifting procedure, where lifting coefficients may only take values 0 or 1. It is well-known that if GUB constraints are present, these can be used to further strengthen cover cuts.

When the knapsack constraint is second-order conic, i.e., $\omega > 0$, cover cuts may still be applied. The process of separation, and extension is in this case however more complicated. Atamtürk and Narayanan (2009) treat the problem of separating and extending cover constraints for second-order conic knapsack constraints, where no GUB constraints are present. Atamtürk et al. (2011) extend this work to the case where GUB constraints are present, and experiment with a number of separation and extension algorithms. Computational results show, that separation and extension of cover inequalities (using GUB constraints) can greatly improve the time needed to solve problems including second-order cone knapsack constraints.

The constraints (9) and (10) define a second-order cone knapsack polytope with the same structure as Z. We separate the conic cover cuts described above using the best performing separation and extension algorithm found in Atamtürk et al. (2011).

5 Solution methodology

We propose to solve the MRCPSP-SNR using a branch-and-cut approach using the commercial solver CPLEX, which solves conic quadratic relaxations in each branch-and-bound node. In

branch-and-cut algorithms, branching typically occurs by fixing a fractional binary variable to 0 in one branch and to 1 in the other. The information derived from such a branching can be used to fix additional variables, through bound and feasibility arguments. For example, fixing the variable $y_{im} := 1$ for some activity i and mode $m \in M_i$ in a node, means that activity i must use mode m in the sub-tree rooted at that node. Using this information, one may find improved lower and upper bounds on the completion time of predecessors and successors of i, and thus fix additional variables. Within the context of CP such a process is known as propagation. In the following we examine different propagation alternatives.

The bound arguments employed to tighten the upper and lower bounds on completion times are based on a so-called Finish-to-Start-Distance (FSD)-matrix, which we will introduce in Section 5.1 along with a number of lower bounding methods used to update the entries of the FSD-matrix.

Following the introduction of the FSD-matrix, we will present the different components of the algorithm: In Section 5.2 we describe a number of preprocessing steps used to reduce the problem size. In section 5.3 we describe how we find initial upper bounds, and how an upper bound can be used to further reduce the problem size. In Section 5.4 we describe how the number of variables of the model may be reduced using lower bound arguments and the FSD-matrix. In Section 5.5, and Section 5.6 we describe how additional variables may be included in order to create a more balanced branch-and-bound tree, and how branching on these variables is propagated. Finally in Section 5.7 and Section 5.8 we describe how additional variables may be fixed, and nodes pruned based on the FSD-matrix.

The approach presented has similarities with the approach used by Zhu et al. (2006) for the MRCPSP, and can be placed within the Constraint Integer Programming (CIP) paradigm (see Achterberg (2007)).

5.1 Finish-to-Start-Distance-matrix and lower bounds

In the following we first give a description of the concept of an FSD-matrix, and then the bounds used in connection with the FSD-matrix.

Finish-to-Start-Distance (FSD)-matrix We employ a slight modification of the FSD-matrix of Zhu et al. (2006), which is in itself a modification of the traditional Start-to-Start-Distance matrix found in the literature, see for instance Bartusch et al. (1988), Brucker and Knust (2000), or Demassey et al. (2005). An FSD-matrix is an integer matrix, $B = (b_{ij})_{\mathcal{A}\times\mathcal{A}}$, which satisfies:

$$\sigma_j - \tau_i \geq b_{ij} + 1, \quad \forall (i,j) \in \mathcal{A} \times \mathcal{A}$$

for all feasible schedules, where τ_i is the completion time of activity i and σ_j is the starting time of activity j. That is, b_{ij} is a lower bound on the amount of time which must pass between the completion time of i and the starting time of j. Note that b_{ij} may be negative, which means that activity j must start before the completion of activity i. Define $\underline{p}_i := \min\{p_{im} | \, m \in \mathcal{M}_i\}$ and $\overline{p}_i := \max\{p_{im} | \, m \in \mathcal{M}_i\}$. Since the relation (5.1) has the following transitive property:

$$\sigma_j - \tau_i \geq b_{ij} + 1 \wedge \sigma_k - \tau_j \geq b_{jk} + 1 \Rightarrow \sigma_k - \tau_i \geq b_{ij} + \underline{p}_j + b_{jk} + 1,$$

the entries of an FSD-matrix may be updated by calculating the transitive closure of B. This can be done using a variant of the Floyd-Warshall algorithm in $O(|\mathcal{A}|^3)$ time, see Zhu et al. (2006). Note that a pair of activities (i,j) must run in parallel if $b_{ij} \geq -\underline{p}_i - \underline{p}_j + 1$ and $b_{ji} \geq -\underline{p}_i - \underline{p}_j + 1$. Given an upper bound, T, on the makespan, the FSD-matrix B may be initialized as follows:

$$b_{ij} = \begin{cases} -\overline{p}_i & \text{if } i = j \\ 0 & \text{if } (i,j) \in \mathcal{E} \\ -T & \text{otherwise,} \end{cases}$$

The difference between the FSD-matrix considered here, and the one of Zhu et al. (2006) is that here the FSD-matrix is defined for all pairs of activities, whereas in Zhu et al. (2006) it is defined only for pairs of activities being related by precedence.

As will be explained in detail in the following sections, an FSD-matrix may be used to eliminate modes and variables from consideration. The effectiveness of the methods rely on good bound arguments for the b_{ij} values. The values b_{ij} may be calculated by constructing the subproblem defined by $\mathcal{S}_i \cap \mathcal{P}_j$ and then applying any lower bounding technique for the RCPSP on this smaller instance. A number of such lower bounds exists in the literature. We do not give a description of these here but instead refer the reader to the excellent work by Klein and Scholl (1999) where 11 such bounds are described and compared. Using the name convention of Klein and Scholl (1999), the bounds considered here are: LB1, LB6, LB8, LB10, and LB11. LB1 is the well-known critical path lower bound, LB6 and LB8 are extensions of the so-called weighed node packing bound by Mingozzi et al. (1998) (see Klein and Scholl (1999)), and LB10 and LB11 are bounds based on evaluating a lower bound for all possible ways of resolving a resource conflict given respectively pairs and triples of activities (again see Klein and Scholl (1999)). The bounds LB6, LB10, and LB11 again use lower bound arguments on subproblems of the problem considered. Note, that this problem may in itself be a subproblem corresponding to a b_{ij} entry. We will refer to the former as *inner* subproblems and the latter as subproblems. When calculating bounds on inner subproblems, one could recursively apply bound arguments, but this can potentially be very time-consuming, and we thus, like Klein and Scholl (1999), only apply the critical path lower bound (LB1) and the capacity lower bound (LB2). We additionally include two MRCPSP-specific lower bounds recently proposed by Muller (2011): The *mode-fixed critical path* (denoted LBX1 in Muller (2011)) is based on calculating the critical path for all possible mode-assignments of a subset of the activities, and the *extended capacity bound* (denoted LB2X in Muller (2011)) is based on the Lagrange Relaxation of the problem of calculating the best possible capacity bound (LB2) taking into account all resources simultaneously.

5.2 Preprocessing

We here describe a number of preprocessing steps, which are applied to the problem initially.

Mode and resource reductions As described by Sprecher et al. (1997) the number of modes and resources may be reduced by application of the following preprocessing procedure: Define $\underline{\tilde{r}}_{ik} := \min\{\tilde{r}_{ikm} | \, m \in \mathcal{M}_i\}$, and $\overline{\tilde{r}}_{ik} := \max\{\tilde{r}_{ikm} | \, m \in \mathcal{M}_i\}$. A mode $m \in \mathcal{M}_i$ is called *non-executable* if either $r_{ikm} > R_k$, for some $k \in \mathcal{R}$, or $\sum_{j \in \mathcal{A} \setminus \{i\}} \underline{\tilde{r}}_{jk} + r_{ikm} > \tilde{R}_k$, for some $k \in \tilde{\mathcal{R}}$. A mode is called *inefficient* if there exits another mode m' of $\mathcal{M}_i$, such that $r_{ikm} \geq r_{ikm'} \, \forall k \in \mathcal{R}$, and $\tilde{r}_{ikm} \geq \tilde{r}_{ikm'} \, \forall k \in \tilde{\mathcal{R}}$. A nonrenewable resource k is called redundant if $\sum_{i \in \mathcal{A}} \overline{\tilde{r}}_{ik} \leq \tilde{R}_k$. Non-executable and inefficient modes, and redundant nonrenewable resources can be removed initially by the use of the algorithm described in Algorithm 1.

Algorithm 1 Pseudo-code for preprocessing of modes and nonrenewable resources

```
Remove non-executable modes.
repeat
  Remove redundant nonrenewable resources.
  Remove inefficient modes.
until No mode removed
```

Resource strengthening The lower bounds LB2, and LB2X depend upon the resource usages. If the resource usage of an activity in a certain mode can be increased without changing the optimal solution, the resulting bounds will be stronger. As in Muller (2011) the following rule is applied after the initial mode and resource reductions: Let $(i, j) \in \mathcal{A} \times \mathcal{A}$ be a pair of activities. We say that two modes $m_i \in \mathcal{M}_i$ and $m_j \in \mathcal{M}_j$ are *incompatible* if they can not be run in parallel, either because of precedence relations between the activities, or because of resource constraints. If a mode $m \in \mathcal{M}_i$ of an activity $i \in \mathcal{A}$, is found incompatible with all other modes of all other activities, then the resource usage is updated as $r_{ikm} := R_k \, , \forall k \in \mathcal{R}$.

5.3 Upper bounds and problem reductions

When solving problems by use of a branch-and-cut approach, good upper bounds are an advantage, as this makes it possible to prune nodes in the search-tree and thus reduce the search space. Additionally, as the number of variables in the CQIP model (6)–(11) is dependant on the earliest and latest possible completion time of each activity, and these completion times again are dependant on the upper bound, a good upper bound both reduces the size of the model, and improves the linear relaxation based lower bound. Further more a good upper bound may be used to deduce new precedence relations and remove modes. In the following we describe how upper bounds are found, how new precedence relations are added to the problem, and how modes are removed from the problem.

Upper bounds To find good upper bounds, we run the Adaptive Large Neighborhood Search (ALNS) algorithm for the MRCPSP proposed by Muller (2011), having changed the evaluation of the nonrenewable resource usages to take into account the square-root term in (9). The algorithm is run using the parameters found in Muller (2011) and with a stopping criteria of 50,000 generated schedules.

Precedence augmentation When an upper bound, UB, is found, one is only interested in finding better solutions. Thus, any sequencing of activities which has a lower bound larger than $UB-1$ can be forbidden. Let the *head*, h_j, of an activity $j \in \mathcal{A}$ be the time that must pass before activity j can be started and let the *tail*, t_j, be the time that must pass from the point where activity j has completed until the project can be completed. The head and tail of an activity j can respectively be read from the entries b_{0j} and b_{jn} of the FSD-matrix. Define $\underline{r}_{ik} := \min\{r_{ikm} | \, m \in \mathcal{M}_i\}$. We now describe how precedence relations may be deduced on the basis of heads and tails.

The following two rules (13) and (14), are a simple generalization to the multi-mode case of a subset of the rules employed by Fleszar and Hindi (2004) in their variable neighborhood search algorithm. Let $i, j \in \mathcal{A}$ be a pair of activities for which no precedence relations exists. Precedence relations may be deduced as follows:

$$h_j + t_i \geq UB \Rightarrow i \rightarrow j \tag{13}$$

$$\exists k \in \mathcal{R} : \underline{r}_{ik} + \underline{r}_{jk} > R_k \wedge h_j + \underline{p}_j + \underline{p}_i + t_i \geq UB \Rightarrow i \rightarrow j. \tag{14}$$

We additionally use the following deduction rule also used by Muller (2011): Let $i, j, k \in \mathcal{A}$ be a triple of activities for which no precedence relation exists and assume that none of the three can be run in parallel. We examine all 6 sequencing possibilities of the three activities: (1) $i \rightarrow j \rightarrow k$, (2) $i \rightarrow k \rightarrow j$, (3) $j \rightarrow i \rightarrow k$, (4) $j \rightarrow k \rightarrow i$, (5) $k \rightarrow i \rightarrow j$, and (6) $k \rightarrow j \rightarrow i$. Given a sequencing, $a \rightarrow b \rightarrow c$, a lower bound on the makespan is $h_a + \underline{p}_a + \underline{p}_b + \underline{p}_c + t_c$. In the following let $LB(1), \ldots, LB(6)$ be such lower bounds corresponding to the sequences (1)–(6). New precedence relations may be deduced as follows:

$$\begin{aligned}
&LB(1) \geq UB \wedge LB(2) \geq UB \wedge LB(5) \geq UB \Rightarrow j \rightarrow i \\
&LB(3) \geq UB \wedge LB(4) \geq UB \wedge LB(6) \geq UB \Rightarrow i \rightarrow j \\
&LB(1) \geq UB \wedge LB(2) \geq UB \wedge LB(3) \geq UB \Rightarrow k \rightarrow i \\
&LB(4) \geq UB \wedge LB(5) \geq UB \wedge LB(6) \geq UB \Rightarrow i \rightarrow k \\
&LB(1) \geq UB \wedge LB(3) \geq UB \wedge LB(4) \geq UB \Rightarrow k \rightarrow j \\
&LB(2) \geq UB \wedge LB(5) \geq UB \wedge LB(6) \geq UB \Rightarrow j \rightarrow k
\end{aligned}$$

If new precedence relations have been added to the problem, the ALNS algorithm is repeated, in order to see if a better upper bound can be found.

Mode diminution As is the case with precedence augmentation, when a new upper bound, UB, is found, only subsequent improvements are of interest. Thus modes, which lead to a lower

bound larger than $UB - 1$ may be removed. When a new UB is found, a mode $m \in M_i$ of an activity $i \in \mathcal{A}$ is removed if one of the following expressions are true:

1. $h_i + p_{im} + t_i \geq UB$.
2. $b_{ii} \geq -p_{im} + 1$.
3. The current entries of the FSD-matrix implies that i must run in parallel with some activity j, and m is incompatible with all modes of j.

The removal of a mode may render other modes non-executable and may render some nonrenewable resources redundant. Thus the two first steps of Algorithm 1 are repeated if a mode is removed. As for precedence augmentation, if a mode is removed, the ALNS algorithm is repeated in order to see if a better upper bound can be found.

5.4 Initial variable reduction

In the branch-and-cut algorithm proposed by Zhu et al. (2006) a so-called variable reduction technique is applied initially in order to fix variables to zero through bound calculations for the entries of the FSD-matrix. We apply a procedure almost identical to the one of Zhu et al. (2006), the only difference being the bound arguments applied. In the variable reduction procedure of Zhu et al. (2006), for each pair of precedence relations $(i, j) \in \mathcal{E}$, the subproject, q, corresponding to the FSD-matrix entry b_{ij} is constructed. Before a potentially time consuming lower bound argument is applied, a genetic algorithm is run on q to see whether a schedule can be found with makespan b_{ij}, if this is the case the bound cannot be improved. The bound arguments applied to q by Zhu et al. (2006) is a truncated branch-and-cut procedure, using a bound based on renewable resources. We instead apply the bound arguments described earlier, i.e., LB1, LB6, LB8, LB10, LB11, LBX1, and LB2X. To further improve the bounds, we also apply a truncated branch-and-cut procedure, but rather than applying the procedure only to the subproblem, we apply it to the complete problem, with the objective of minimizing the term

$$\sigma_i - \tau_j = \sum_{m \in \mathcal{M}_i} \sum_{t=\mathcal{T}^e_{im}}^{\mathcal{T}^l_{im}} x_{itm}(t - p_{im}) - \sum_{m \in \mathcal{M}_j} \sum_{t=\mathcal{T}^e_{jm}}^{\mathcal{T}^l_{jm}} t \cdot x_{jtm},$$

Including all activities of the project may produce better bounds, than only considering activities of the subproject, but as the problem is larger, solving it may also be more time consuming. In order to save time we, as Zhu et al. (2006), only apply the bound arguments to a subset of the subprojects. Zhu et al. (2006) only apply the bound arguments if their heuristic can not find a solution with a makespan equal to the current value of b_{ij}, which is sensible since only in this case may the bound be improved. In our case it may be possible to improve the bound regardless of such a check, because the truncated branch-and-cut procedure considers all activities of the project. Even so, we choose to run the truncated branch-and-cut procedure only if the heuristic (in our case the ALNS algorithm) can not find a solution equal to the current value of b_{ij}, as we deem such a case the most promising for improving the bound b_{ij}.

In addition to removing variables before starting the branch-and-cut algorithm, variable reduction may also improve the bounds of the FSD-matrix (because of the truncated branch-and-cut procedure), which is a benefit as the FSD-matrix, as we shall see, is used throughout the branch-and-bound tree.

5.5 Variables and branching

Branching based directly on fractional x-variables has the potential to create an unbalanced search-tree, as the branch corresponding to $x_{jtm} = 0$ may not change the value of the LP-relaxation much. We now describe two methods, which may produce a more balanced search-tree. We will in Section 6 examine whether it is profitable from a computational point of view to include these methods.

Mode-branching (y-variables) One solution for creating a more balanced search-tree is branching on the modes of an activity, i.e., $\sum_{t=\mathcal{T}^e_{jm}}^{\mathcal{T}^l_{jm}} x_{jtm} = 0$ in one branch, and $\sum_{t=\mathcal{T}^e_{jm}}^{\mathcal{T}^l_{jm}} x_{jtm} = 1$ in the other, for some activity $i \in \mathcal{A}$ and mode $m \in \mathcal{M}_i$. In order to achieve this behavior, we add the artificial variables $y_{im} := \sum_{t=\mathcal{T}^e_{im}}^{\mathcal{T}^l_{im}} x_{jtm}$ described earlier explicitly to the model, and let CPLEX decide when to branch on these variables.

Time-branching (SOS) As described by Zhu et al. (2006), the x-variables, $\{x_{jtm} : m \in \mathcal{M}_j, t = \mathcal{T}^e_{jm}, \ldots, \mathcal{T}^l_{jm}\} \forall j \in \mathcal{A}$, divide nicely into Special Ordered Sets (SOS) of type 1. Each variable x_{jtm} is assigned a weight t. Let x^* be a fractional solution in some branch-node. CPLEX branches on a SOS, S, of some activity j by dividing the variables of S into two sets: S_1 and S_2, and in one branch setting $\sum_{(t,m)\in S_1} x_{jtm} \leq 0$, while in the other branch setting $\sum_{(t,m)\in S_2} x_{jtm} \leq 0$. The division is such that $\forall (t,m) \in S_1 : tx^*_{jtm} \leq C$ and $\forall (t,m) \in S_2 : tx^*_{jtm} \geq C$, where $C = \sum_{i \in S} tx^*_{jtm}$. Thus branching on a SOS can be seen as branching on the time-interval, such that in one branch the activity j must complete no later than $\lfloor C \rfloor$, while in the other branch the activity j may complete no earlier than $\lceil C \rceil$.

5.6 Variables and propagation

As mentioned earlier, when branching on a variable, additional information may be deduced. This information may be used to update the FSD-matrix, and subsequent calculation of the closure of the FSD-matrix may lead to a tightening of the earliest and latest possible completion of some activities, and the corresponding variables may be fixed to zero. The information contained in the FSD-matrix of the current branch-and-bound node is passed down to its children nodes, so information gathered from branching decisions are accumulated as branching progresses. Fixing variables may result in improved lower bounds, but propagation may be time-consuming and there is a trade-off between the time spent per branch-and-bound node, and the gain in number of branch-and-bound nodes having to be considered. We will in Section 6 be examine whether it is profitable to include propagation. In the following we describe how propagation is performed for each of the three possible branching types, that is, branching on x-varibles, branching on y-variables, and branching on Sepcial Ordered Sets.

y-variables When mode-branching occurs, one disallows a mode m, for an activity j in one branch ($y_{jm} = 0$), while fixing the activity to run in that mode in the other branch ($y_{jm} = 1$). Since some of the bound arguments used to calculate the entries of the FSD-matrix are dependant on the available modes, reapplying these bound arguments may lead to improvements of some entries of the FSD-matrix.

SOS When time-branching occurs, depending on the branch, some activity j is fixed to complete either before or after some point in time τ. This may again be reflected in the entries of the FSD-matrix: requiring that j completes before time τ the FSD-matrix may be updated as $b_{j0} = -(\tau + 1)$, while requiring that j completes after time τ the FSD-matrix may be updated as $b_{0j} = \tau - \max\{p_{jm} | m \in \mathcal{M}_j\} + 1$.

x-variables When branching on x-variables, in one branch some activity j is fixed to complete at some point in time τ using some mode m, while in the other branch j may not complete at time τ using mode m.

In case of the first branch, the FSD-matrix updated firstly in the same manner as for the y-variables, i.e., reapplying the bound arguments that depend on available modes, and secondly by setting $b_{j0} = -(\tau + 1)$ and $b_{0j} = \tau - p_{jm} + 1$. This update imply $\sigma_j + p_{jm} - 1 = \tau$.

In case of the second branch, i.e, the $x_{j\tau m} = 0$ branch, let $\underline{\tau} = \min\{t \in \mathcal{T} : x_{jtm} = 0, m \in \mathcal{M}_j\}$ and let $\overline{\tau} = \max\{t \in \mathcal{T} : x_{jtm} = 0, m \in \mathcal{M}_j\}$, i.e., the earliest and latest possible completion time

given the current state of the branch-and-bound tree. The FSD-matrix is then updated by setting $b_{j0} = -(\underline{\tau} + 1)$ and $b_{0j} = \overline{\tau} - \max\{p_{jm} | m \in \mathcal{M}_j\} + 1$.

5.7 FSD-matrix-based mode removal

Based on the entries of the updated FSD-matrix and the modes disallowed at the current node, further modes may be removed:

Heads and tails If an activity $i \in \mathcal{A}$ and mode $m \in \mathcal{M}_j$ exists, such that $h_i + p_{im} + t_i \geq UB$, the mode may be removed (this check is equivalent to check 1 from mode diminution).

Parallel activities If the current entries of the FSD-matrix implies i must run in parallel with some activity j, and m is incompatible with all modes of j, then the mode may be removed (this check is equivalent to check 3 from mode diminution).

Nonrenewable resources If an activity $i \in \mathcal{A}$ and mode $m \in \mathcal{M}_j$ exists such that m is non-executable, then the mode may be removed.

5.8 FSD-matrix-based pruning

The entries of the FSD-matrix and the modes currently disallowed may also be used to prune search nodes:

Infeasible completion times If through the tightening of bounds on earliest and latest completion times, a variable, which was earlier fixed to one by a branching, has to be fixed to zero, the node may be pruned.

Infeasible modes If through the application of the mode removal arguments just described, a mode, which was earlier fixed to one by a branching, has to be fixed to zero, the node may be pruned.

Parallel activities If the current entries of the FSD-matrix implies i and j must run in parallel and no pair of modes is compatible, or if the FSD-matrix implies that the activities i, j, and k must run in parallel and no triple of modes is compatible, then the node may be pruned.

6 Computational experiments

In this section we describe the computation experiments performed. This includes testing the effect of the variables included in the model, the effect of cutting, and the effect of branching-based propagation. We conclude with a study of the effect of the risk factor on the average makespans, and on the time taken to find solutions. The algorithm has been coded in C++, compiled with gcc 4.4.3 and the experiments have been run on a PC with 2 Intel(R) Xeon(R) CPU X5550 @ 2.67GHz (16 cores in total, but only a single core is used), with 24 GB of RAM, and running Ubuntu 10.4. CPLEX is left at its default settings, except for the following parameters: GUB-covers are set to be separated aggressively, clique-covers are set to be separated very aggressively, the local-branching heuristic is on and the cutfactor is set to 6. These settings were taken from Zhu et al. (2007). For the root node, the emphasis is set to best bound, after which it is set to balanced as this gave the best performance for preliminary tests. Only a single thread is used. For the truncated branch-and-cut algorithm used in the variable reduction procedure ,CPLEX is left at its default settings, except for the following parameters: GUB-covers are set to be separated aggressively, clique-covers are set to be separated very aggressively, the emphasis is set to best bound, repeat presolve is set to off, the node limit is set to 100, and the time limit is set to 5 seconds. Again only a single thread is used. The code is available for download at `http://diku.dk/~laurent`.

6.1 Benchmark instances

As the Single-mode Resource-Constrained Project Scheduling Problem (SRCPSP) has not been considered before, there are no benchmark instances available in the literature. There are, however, a number of benchmark instances for the MRCPSP available from the PSPLIB, which have been widely used in the literature (see also Kolisch and Sprecher (1997)). The PSPLIB may be found at `http://129.187.106.231/psplib/`. It seems natural to extend these benchmark instances to the problem considered here. We extend the benchmark classes: J10, J12, J14, J16, J18, and J20. These benchmark classes consists of instances containing respectively 10, 12, 14, 16, 18 and 20 non-dummy activities. Each activity may be performed in up to 3 different modes, there are two nonrenewable, and two renewable resources. The number of instances in each benchmark class J10–J20 is respectively 536, 547, 551, 550, 552, and 554. For these, all optimal solutions are known. The new instances are generated as follows: the mean of the stochastic nonrenewable resource usage ($\tilde{r}_{jkm}$) is set to the value of the nonrenewable resource usage of the corresponding deterministic instance, and the standard deviation (σ_{jkm}) is chosen at random within the range $[a \cdot \tilde{r}_{jkm}; b \cdot \tilde{r}_{jkm}]$, where $a = 0$, and $b = 0.15$. The value of the risk factor ϵ, is not part of the instance. The instances are available for download at `http://diku.dk/~laurent`.

6.2 Evaluation of components

We here examine the different components of the algorithm, in order to establish which combinations perform the best. As the instances may take a long time to solve, we restrict these experiments to the same subset of instances from the J20 benchmark class as Zhu et al. (2006): J2013 containing 10 instances.

Model (no propagation) We first examine the effect of including the redundant y-variables, and the Special Ordered Sets when solving the model without propagation. All other components are disabled, i.e., variable reduction, cutting, and propagation. The results can be see in Table 1. Including y-variables has a slightly positive effect on the integrality gap compared to the model with no y-variables and SOS. All other combinations are worse.

Table 1: Effect of including the redundant y-variables and the special ordered sets. The column **#Opt** is the number of instances solved to optimality within 3600 seconds, the column **Dev. Best** is average percentage deviation from the optimal solution for the deterministic case, the column **Gap** is the average percentage integrality gap of instances not solved to optimality, the column **Nodes** is the average number of nodes in the branch-and-bound tree, and the column **Time** is the total time used across all 10 instances. In the **Vars.** column y means the y-variables are included, $\underline{y}$ means the y-variables are given priority over x-variables for branching, and *sos* means that SOS are included.

Vars.	#Opt	Dev. Best(%)	Gap(%)	Nodes	Time(s)
	6	3.38	6.064	34066	16396
sos	3	3.94	11.721	138742	26543
y	6	2.81	4.289	12816	17391
$\underline{y}$	5	3.15	13.645	23437	21302
y,*sos*	3	4.23	12.205	65189	26784
$\underline{y}$, *sos*	1	4.23	21.014	52707	32434

Variable reduction (no propagation) We next examine the effect of initial variable reduction. We use the best combination from the previous test, i.e., y-variables are included. Again all other components are disabled. The results can be seen in Table 2. Variable reduction has a slight negative effect, and many more branch nodes are examined.

Table 2: Effect of variable reduction. The column **Vars.** is the average number of binary variables in the problem, and the column **Rem.** is the average number of varibles removed by variable reduction. The remaning columns are as earlier.

	#Opt	**Dev. Best(%)**	**Gap(%)**	**Nodes**	**Time(s)**	**Vars.**	**Rem.**
No Var. Red	6	2.81	4.289	12823	17390	0.00	0.00
Var. Red	6	2.88	5.440	26278	18578	906.25	63.75

Cuts We next examine the effect of cutting on the stochastic nonrenewable resource constraints. y-variables are included, and variable reduction is disabled, as are the remaining components. The results can be seen in Table 3. Cutting locally throughout the branch-and-bound tree performs the best. This agrees with the findings of Atamtürk et al. (2011), where cutting locally throughout the branch-and-bound tree also showed the best performance. Disappointingly, the effect of cutting is slight. We beleive the reason for this is, that there are only 2 second-order cone constraints in the problem, and that these may thus not influence the complete problem structure much.

Table 3: Effect of including cuts for the nonrenewable resources. The columns are as earlier.

	#Opt	**Dev. Best(%)**	**Gap(%)**	**Nodes**	**Time(s)**
No cutting	6	2.81	4.289	12825	17389
Root cutting	6	2.88	4.951	14794	18540
Global cutting	6	3.10	6.732	12692	17470
Local cutting	6	2.59	4.618	14314	17867

Model (with propagation) We now turn our attention to propagation. Propagating on a branching type takes time, and we want to examine, which combination of variables included and which type of branch propagated leads to the best performance. Recall that we may propagate on any of the following: x-variables, y-variables, and Special Ordered Sets. For these experiments cutting is enabled, as is FSD-matrix-based mode removal and pruning, while variable reduction is disabled. The results can be seen in Table 4. In order to better present the differences between combinations, the table is sorted with respect to the number of optimal solutions, on ties the average deviation from the optimal solutions known from the deterministic case, on ties finaly on the average gap. The combination where branching on y-variables is propagated, and y-variables are given priority over x-variables for branching gives the best performance solving a total of 8 instances to optimality, which is better than the 6 instance solved to optimality without propagation.

Variable reduction (with propagation) As mentioned earlier, variable reduction may improve the bounds of the FSD-matrix and as the FSD-matrix is used as part of the propagation, it is of interest to examine the effect of enabling variable reduction, when propagation is used. For these experiments the best propagation strategy from the previous test is used, and some components are enabled. The results can be seen in Table 5. Variable reduction can be seen to improve the average deviation from the optimal solutions known from the deterministic case, i.e., the average makespan, but the integrality gap is worsened.

Summary Table 6 summarise the improvement of using the best solution strategy found compared to CPLEX.

Table 4: Performance for different combinations of variables and type of branching propagated. An x, y and *sos* in the **Prop** column means that branching on x-variables, y-variables, and Special Ordered Sets respectively are propagated. The remaining columns are as earlier.

Vars.	**Vars. prop.**	**#Opt**	**Dev. Best(%)**	**Gap(%)**	**Nodes**	**Time(s)**
$\underline{y}$	y	8	2.59	4.417	15020	15996
$\underline{y}$	x,y	8	2.59	4.963	14371	15007
y	x,y	7	3.16	7.303	18254	15528
$\underline{y}$	x	6	3.10	8.959	11275	19137
y	x	6	3.16	5.922	20319	17514
y	y	6	3.38	6.064	13743	16708
$\underline{y}$,*sos*	y,*sos*	5	3.65	15.080	25406	20497
$\underline{y}$,*sos*	y	5	3.65	15.111	27335	22059
$\underline{y}$,*sos*	x,y,*sos*	5	3.65	15.227	24348	21197
$\underline{y}$,*sos*	x,y	5	3.65	15.252	24007	20761
y,*sos*	x,y,*sos*	4	3.65	9.890	43433	25845
y,*sos*	x,*sos*	4	3.65	10.299	47898	25867
sos	x,*sos*	4	3.65	10.497	30682	25519
y,*sos*	y	4	3.67	10.355	41874	25867
	x	4	3.94	7.391	28995	22852
sos	x	4	4.00	13.137	31201	25568
y,*sos*	*sos*	4	4.23	10.832	49584	26229
$\underline{y}$,*sos*	*sos*	3	3.65	17.719	34299	29359
y,*sos*	y,*sos*	3	3.94	9.164	49029	27308
y,*sos*	x,y	3	3.94	9.513	39935	26958
y,*sos*	x	3	3.94	10.487	43893	26933
sos	*sos*	3	4.23	11.048	37344	26750
$\underline{y}$,*sos*	x,*sos*	2	4.23	18.585	29066	29404
$\underline{y}$,*sos*	x	2	4.23	18.690	28547	29416

Table 5: Effect of including variable reduction with propagation. The columns are as earlier.

	#Opt	**Dev. Best(%)**	**Gap(%)**	**Nodes**	**Time(s)**	**Vars.**	**Rem.**
No Var. Red.	8	2.59	4.350	15321	15471	0.00	0.00
Var. Red.	8	2.22	5.999	12702	14823	906.25	63.38

6.3 Final results

We here examine the effect of the risk factor ϵ on the average makespan. We use the best solution strategy found in the previous experiments, that is, y-variables are included and given priority over x-variables for branching, cutting is performed locally in each branch-and-bound node, and variable reduction is used. For these experiments we run on all instances from the J10–J20 benchmark classes. We remind the reader that the number of instances in each benchmark classes J10–J20 is respectively 536, 547, 551, 550, 552, and 554. We experiment with four values of ϵ shown in Table 7. Recall that for the benchmark instances considered $|\tilde{\mathcal{R}}| = 2$, and thus $\epsilon_k = \sqrt{\epsilon}, \forall k \in \tilde{\mathcal{R}}$. Setting $\epsilon = 0$ corresponds to not taking uncertainty into account. For this case the problem is a regular MIP rather than a CQIP and corresponds to the deterministic case.

The results from running on the J10–J20 benchmark instances can be seen in Table 8. As expected the average makespan of the solutions increase as the value of ϵ increases, illustrating the trade-off between the length of the makespan and the probability that the project will not run "over budget". The increase in makespan is around 7% when requiring the project to stay on budget with a 99% probability as compared to not taking stochasticity into account. This

Table 6: Comparison of best solution strategy to CPLEX.

	#Opt	**CP Dev.(%)**	**Gap(%)**	**Nodes**	**Time(s)**
CPLEX	6	3.38	6.064	34066	16396
Best	8	2.22	5.999	12702	14823

Table 7: Values of ϵ used, and the corresponding values of ϵ_k and $\Phi^{-1}(\epsilon_k)$

ϵ	ϵ_k	$\Phi^{-1}(\epsilon_k)$
0.99	0.995	2.58
0.95	0.975	1.96
0.85	0.922	1.42
0	0	0

increase seems an acceptable price to pay for robustness. When the value of ϵ increases, so does the number of infeasible problem instances, but this number stays relatively low: up to around 2% of the instances for J12–J20, and up to around 10% for J10.

On all instances, when increasing ϵ to a positive number, i.e., changing the model from being a regular MIP model to being a CQIP, the computational time increases. The average times per instance are however relatively small, and using less than 5 minutes to solve a scheduling problem, which could span days, weeks, or months seems acceptable. When comparing the minimum, maximum, and average time, one observes that there is a big variance, with some instances taking almost no time, while others take the full time allotted (the reason why some run times are above the allotted 3600 seconds, is that the time spend cutting and solving in the root node is not counted towards the 3600 seconds). This indicates that a few instances are hard taking up a lot of time, while the remaining are solved relatively fast.

We can consistently solve almost all instances up to J18 (1 is unknown for J16 and $\epsilon = 0.95$, and up to 4 are unknown for J18), while for J20 around 20–25 instances are not solved to optimality. For J20 the average optimality gap is relatively small (below around 8%), while it is somewhat larger for J18 (below around 16%). The heuristic performs remarkably well and is less than 1% from the best found solution, and in some cases much closer.

7 Conclusion

We have presented a new variant of the MRCPSP, where nonrenewable resources are stochastic and described by a mean and a variance, and the nonrenewable resources take the form of chance constraints. We have shown how the problem can be modelled as a CQIP, where the chance constraints are modelled as second-order cone constraints. This problem is of interest both in itself, and as a stepping stone towards a completely stochastic RCPSP model.

In order to solve the problem we have proposed a branch-and-cut algorithm and experimented with propagation rules based on the branching decisions performed in the branch-and-bound tree, and with cuts for the nonrenewable resources. These experiments show that the branch-and-cut algorithm using propagation outperforms CPLEX.

We have further experimented with the branch-and-cut algorithm on a large number of benchmark instances found in the literature and adapted to the MRCPSP-SNR. These experiments show that taking stochasticity into account only results in an increase of around 7% of the average makespan. Solving a CQIP instead of a MIP as expected results in an increase of the running time, but on average these running times stay relatively low.

It would be of interest to investigate how to merge stochastic nonrenewable resources, with stochastic models for the RCPSP such as stochastic activity durations, or renewable resource consumptions.

Table 8: Results from running on the J10–J20 benchmark classes. The column **#O** is the number of instances solved to optimality, **#I** is the number of instances proved to be infeasible, **#U** is the number of instances which were neither proved optimal optimal nor infeasible, **Mks** is the average makespan, **ΔH** is the average percentage deviation between the final solution and the solution found by the heuristic, **Gap** is the average gap in percent when the algorithm terminated, **Avg**, **Min** and **Max** is respectively the average, minimum, and maximum time in seconds per instance

	ϵ	**#O**	**#I**	**#U.**	**Mks**	**ΔH(%)**	**Gap(%)**	**Min(s)**	**Max(s)**	**Avg(s)**
J10	0.99	482	54	0	20.39	0.01	0.000	0.00	232.50	3.65
	0.95	507	29	0	20.52	0.01	0.000	0.00	124.05	2.78
	0.85	526	10	0	20.36	0.03	0.000	0.00	372.57	3.96
	0	536	0	0	19.04	0.00	0.000	0.00	63.61	1.53
J12	0.99	536	11	0	22.95	0.16	0.000	0.00	2799.68	16.24
	0.95	542	5	0	22.59	0.10	0.000	0.00	1092.07	7.73
	0.85	544	3	0	22.24	0.09	0.000	0.00	466.30	6.03
	0	547	0	0	21.35	0.03	0.000	0.00	236.34	3.04
J14	0.99	542	9	0	24.95	0.32	0.000	0.00	1825.06	22.67
	0.95	546	5	0	24.51	0.29	0.000	0.00	828.58	17.23
	0.85	546	5	0	24.08	0.30	0.000	0.00	2248.01	17.91
	0	551	0	0	23.23	0.06	0.000	0.00	324.54	4.02
J16	0.99	544	6	0	26.61	0.55	0.000	0.00	3017.48	38.39
	0.95	543	6	1	26.09	0.51	5.263	0.00	3769.63	41.63
	0.85	545	5	0	25.79	0.46	0.000	0.00	3987.37	32.00
	0	550	0	0	25.00	0.14	0.000	0.00	3058.17	13.86
J18	0.99	546	2	4	28.23	0.84	11.420	0.00	4137.56	121.19
	0.95	548	2	2	27.79	0.73	13.255	0.00	4170.98	98.67
	0.85	549	2	1	27.39	0.67	15.634	0.00	3998.94	77.36
	0	552	0	0	26.66	0.29	0.000	0.00	2186.09	26.89
J20	0.99	524	5	25	29.87	0.97	8.432	0.00	4649.16	289.04
	0.95	530	3	21	29.31	0.97	8.169	0.01	5171.45	260.19
	0.85	530	2	22	28.87	1.04	8.476	0.01	4842.29	253.77
	0	548	0	6	27.88	0.43	8.063	0.01	4031.13	95.22

References

Achterberg, T. *Constraint Integer Programming*. PhD thesis, Universitätsbibliothek, 2007.

Atamtürk, A., Narayanan, V. The submodular 0-1 knapsack polytope. *Discrete Optimization*, 6: 333–344, 2009.

Atamtürk, A., Muller, L. F., Pisinger, D. Separation and extension of cover inequalities for second-order conic knapsack constraints with generalized upper bounds. Technical report, Department of Management Engineering, Technical University of Denmark, Denmark, 2011.

Bartusch, M., Möhring, R., Radermacher, F. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988.

Blażewicz, J., Lenstra, J., Kan, A. R. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

Boyd, S., Vandenberghe, L. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787.

Brucker, P., Knust, S. A linear programming and constraint propagation-based lower bound for the rcpsp. *European Journal of Operational Research*, 127(2):355–362, 2000.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

Chtourou, H., Haouari, M. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & Industrial Engineering*, 55(1):183 – 194, 2008.

Demassey, S., Artigues, C., Michelon, P. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *Informs Journal on Computing*, 17:52–65, 2005.

Fernandez, A., Armacost, R. The role of the nonanticipativity constraint in commercial software for stochastic project scheduling. *Computers & Industrial Engineering*, 31(1-2):233–236, 1996.

Fernandez, A., Armacost, R., Pet-Edwards, J. A model for the resource constrained project scheduling problem with stochastic task durations. In *7th Industrial Engineering Research Conference Proceedings*, 1998a.

Fernandez, A., Armacost, R., Pet-Edwards, J. Understanding simulation solutions to resource constrained project scheduling problems with stochastic task durations. *Engineering Management Journal*, 10:5–14, 1998b.

Fleszar, K., Hindi, K. S. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155(2):402–413, 2004.

Golenko-Ginzburg, D., Gonik, A. Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics*, 48(1):29–37, 1997.

Hartmann, S., Briskorn, D. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

Igelmund, G., Radermacher, F. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1):29–48, 1983a.

Igelmund, G., Radermacher, F. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13(1):1–28, 1983b.

Klein, R., Scholl, A. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112:322–346, 1999.

Kolisch, R., Sprecher, A. Psplib – a project scheduling library. *European Journal of Operational Research*, 96:205–216, 1997.

Lambrechts, O., Demeulemeester, E., Herroelen, W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, 11 (2):121–136, 2008a.

Lambrechts, O., Demeulemeester, E., Herroelen, W. A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*, 111(2):493–508, 2008b.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

Möhring, R. H., Radermacher, F. J. Introduction to stochastic scheduling problems. In Neumann, K., Pallaschke, D., editors, *Contributions to Operations Research, Proceedings of the Oberwolfach Conference*, volume 240 of *Lecture Notes in Economics and Mathematical Systems*, pages 72–130. Springer-Verlag, 1985.

Möhring, R., Radermacher, F., Weiss, G. Stochastic scheduling problems igeneral strategies. *Mathematical Methods of Operations Research*, 28(7):193–260, 1984.

Möhring, R., Radermacher, F., Weiss, G. Stochastic scheduling problems ii-set strategies. *Mathematical Methods of Operations Research*, 29(3):65–104, 1985.

Muller, L. F. An adaptive large neighborhood search algorithm for the multi-mode resouce-constrained project scheduling problem. Technical report, Department of Management Engineering, Technical University of Denmark, Denmark, 2011.

Radermacher, F. Cost-dependent essential systems of es-strategies for stochastic scheduling problems. *Methods of Operations Research*, 42:17–31, 1981.

Radermacher, F. Analytical vs. combinatorial characterizations of well-behaved strategies in stochastic scheduling. *Methods of Operations Research*, 53:467–475, 1986.

Sprecher, A., Hartmann, S., Drexl, A. An exact algorithm for project scheduling with multiple modes. *OR Spectrum*, 19(3):195–203, 1997.

Stork, F. *Stochastic Resource Constrained Project Scheduling.* PhD thesis, TU-Berlin, 2001.

Tsai, Y., D Gemmill, D. Using tabu search to schedule activities of stochastic resource-constrained projects. *European Journal of Operational Research*, 111(1):129–141, 1998.

Valls, V., Laguna, M., Lino, P., Pérez, A., Quinatanilla, S. Project scheduling with stochastic activity interruptions. In Weglarz, J., editor, *Project scheduling: recent models, algorithms, and applications*, pages 333–353. Kluwer, Amsterdam, 1998.

Van de Vonder, S. *Proactive-reactive procedures for robust project scheduling.* PhD thesis, Katholieke Universiteit Leuven, 2006.

Van de Vonder, S., Demeulemeester, E., Leus, R., Herroelen, W. Proactive-reactive project scheduling-trade-offs and procedures. *International Series in Operations Research and Management Science*, 92:25, 2006.

Van de Vonder, S., Demeulemeester, E., Herroelen, W., Leus, R. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97(2):227 – 240, 2005.

Van de Vonder, S., Ballestn, F., Demeulemeester, E., Herroelen, W. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, 52(1):11 – 28, 2007a.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10(3):195–207, June 2007b.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189 (3):723 – 733, 2008.

Zhu, G., Bard, J., Yu, G. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing*, 18(3):377, 2006.

Zhu, G., Bard, J., Yu, G. A two-stage stochastic programming approach for project planning with uncertain activity durations. *Journal of Scheduling*, 10(3):167–180, June 2007.

Many processes within production scheduling and project management involve the scheduling of a number of activities, each activity having a certain duration and requiring a certain amount of limited resources. The duration and resource requirements of activities are commonly the result of estimations, and thus generally subject to uncertainty. If this uncertainty is not taken into account the resulting schedules may not be robust in the sense that, when executed, the uncertainty may cause the schedules to take longer than expected, consume moreresources, or be outright infeasible. We propose a new variant of the Multi-mode Resource-Constrained Project Scheduling Problem, where the nonrenewable resource requirements of each mode is given by a Gaussian distribution, and the nonrenewable resource constraints must be satisfied with a certain probability p. Such constraints are also known as chance constraints. We present a Conic Quadratic Integer Program model of the problem, and describe and experiment with a branch-and-cut algorithm for solving the problem. In each node of thebranch-and-bound tree, the branching decisions are propagated in order to remove variables from the problem, and thus improve bounds. In addition we experiment with cutting on the conic quadratic resource constraints. Computational experiments show that the branch-and-cut algorithm outperforms CPLEX 12.1. We finally examine the "cost of uncertainty" by investigating the relation between values of p, the makespan, and the solution time.

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk